LLNL-CONF-410976

Dataflow System: Is It A Viable Scalable Graph Analysis Engine?

A. Yoo

March 2, 2009

International Conference on Parallel Processing
Vienna, Austria
September 22, 2009 through September 25, 2009

Dataflow System: Is It A Viable Scalable Graph Analysis Engine?

Andy Yoo

Lawrence Livermore National Laboratory
Livermore, CA 94551

Abstract

The feasibility of using dataflow systems for running complex graph queries is studied in this paper. A general query optimization framework for parallel dataflow systems is also proposed. The proposed methods are used to optimize a suite of benchmark queries, and their effectiveness is evaluated. The performance of the optimized queries is measured on an actual parallel dataflow machine using a large semantic graph and compared to that of equivalent SQL queries on a high-end parallel relational database system. The study has revealed that dataflow system can achieve significant performance improvement over state-of-art database systems and can be a viable and scalable alternative to run large complex graph queries.

1 Introduction

Semantic graph analysis has become an increasingly important problem in recent years and plays a vital role in counter-intelligence, bioinformatics, business intelligence, and web mining, and so on. A semantic graph consists of typed vertices such as `person` and `organization` and typed edges that represent the relationships between the vertices such as `works_for` and `visit`. Vertices and edges also have attributes associated with them (e.g., *name* and *date_of_visit*).

Semantic graphs, which are usually formed by fusing fragmental information obtained from many different sources like web documents, news articles, and public records, have very complex structures. The semantic graphs also tend to continue to grow in size, and it is not uncommon for scientists and knowledge engineers to deal with semantic graphs with billions of vertices and edges in practice. The large size and complexity of the semantic graphs, unfortunately, make the semantic graph analysis an inherently difficult problem. Furthermore, many of the queries performed on real-world semantic graphs are very complicated with high-order complexity and generate a large volume of intermediate results [10, 19, 17, 16, 14, 15, 4]. This makes analyzing large semantic graphs in a scalable and efficient manner even more challenging.

Graph data is commonly stored in conventional relational databases mainly due to their availability and ease of use. However, using relational databases to run graph analysis applications can have significant impact on the query performance, since the complicated graph queries often have to be implemented using a set of expensive SQL operations. Their inability to scale becomes evident when the graph queries are executed on very large graph data [10]. In addition, the SQL query performance is hard to optimize, because how an SQL code is translated into underlying primitives is largely opaque to users and varies widely from one SQL compiler to another. These limitations of the relational databases as a graph analysis engine warrant investigation of alternative systems for the scalable semantic graph analysis.

Dataflow, whose inception dates back to the sixties [8, 11], is a simple and powerful model of parallel computation and has been widely used in many areas of computer science, including programming languages, computer architecture, and signal processing [13]. In dataflow architecture [6, 7], there is no single locus of control and the computation is driven by the availability of data, naturally enabling asynchronous data parallelism. The simplicity and inherent data parallelism of the dataflow model has prompted the use of the model to solve a wide range of web applications in recent years [5].

Motivated by the advantages of the dataflow model, we investigate the potential use of dataflow systems as a platform to run complex queries on large graphs. We first propose a set of techniques that can be generally applied to optimize the performance of complex graph queries. These techniques can be collectively used without any negative performance effect and therefore enable automatic query optimization, which can be particularly

Prepared by LLNL under Contract DE-AC52-07NA27344.

useful when a graph query language [3, 9] is used to write user queries. Also, these optimization methods can be used as a set of guidelines to write efficient and scalable graph analysis applications.

The performance of graph benchmark queries [10, 21], which are optimized by applying the proposed techniques, is then measured on an actual dataflow machine using a large real-world graph and compared to that of equivalent queries on state-of-art SQL machines. The performance study shows that dataflow system that optimizes its performance by taking advantage of the inherent data parallelism of the dataflow model and offers a large degree of flexibility for users to optimize their queries can achieve significant performance improvement over the state-of-art parallel relational database systems and serve as a viable platform for the scalable graph analysis.

The paper is organized as follows. Section 2 describes a dataflow system evaluated in this research, and the proposed optimization techniques are presented in in Section 3. The results from performance study are discussed in Section 4, followed by concluding remarks in Section 5.

2 Dataflow for Large Graph Analysis and Testbed

As stated earlier, conventional database processing technologies are not suitable for ingesting and analyzing complex, massive semantic graphs. It was also shown that the conventional relational databases used as graph analysis engine do not scale even for graphs of moderate size [10]. The limited scalability is inherent in the implementation of the relational database management systems architecture that was originally designed for transaction processing.

Dataflow is a simple and powerful model that is believed to offer a basis for a scalable architecture to replace the relational databases for graph analysis. In dataflow model, there is no notion of a single locus of control. The dataflow model describes computation in terms of locally controlled events where each event corresponds to the firing of an actor [13]. An actor can be considered as a function that processes the input data. An actor fires and starts its operation when all the inputs it requires become available. In a dataflow execution, many actors usually fire simultaneously, capacitating data parallelism. This inherent data parallelism facilitates the development of parallel data management system that can process different portions of data simultaneously in the most scalable and efficient manner. In addition, the dataflow model provides users with a large degree of flexibility that allows them to perform low-level query optimization, which is largely left to the SQL compiler in the relational database systems.

In this research, we rely on a dataflow system called *data analytics supercomputer* (DAS) [12] as a testbed. DAS is a dataflow system based on the *active disk* architecture [18]. Regardless of the algorithm used or the entity being processed in a semantic graph analysis, the analysis requires that all the records be compared to each and every one of the other records in worst case. There are basically two approaches to solve this problem. In a typical n-tier architectures, the data is stored at a relational database system and is retrieved to the middle-tier on which the actual computation is performed. This approach does not scale for large data sets as moving a large amount of data can be significant performance bottleneck. The DAS system, on the other hand, brings the computation to the data that is stored on a distributed parallel computer.

Hardware-wise, the DAS system can operate virtually on any commodity clusters. The testbed at LLNL consists of 40 Sun Fire X2100 servers, where each node in the testbed has a dual-core AMD Opteron processor running on 2.6 GHz with 4 GB of memory and 400 GB of disk space. The DAS boosts the performance via using innovative software that controls multiple nodes so that they act as one, hence, optimizing computations. User queries are implemented in a custom dataflow definition language, which simplifies the complexity associated with parallel processing by allowing applications to be implicitly executed on data in parallel. The user queries are precompiled before being distributed to the individual nodes for execution, to reduce inefficiencies and increase processing speed and scalability. The details of the design and usage of the system is further articulated in [1, 2].

3 Optimization Techniques for Parallel Dataflow Systems

One of the main goals of this research is to develop query optimization framework that can be generally applied to the optimization of user queries, especially graph analysis queries, to boost their performance on dataflow systems. The proposed optimization methods are described in this section. The contributions from these methods are two-fold. First, any combinations of these methods can be collectively used to optimize queries without any interference from each other that may cause negative performance effect, since they are complementary and mutually exclusive. A related advantage of this property is that the query optimization process can be easily automated by applying the proposed optimization techniques algorithmically. This automatic query optimization capability would be particularly useful when user queries are constructed using a graph query language [3, 9], in which the queries are optimized by a compiler. Second, the proposed optimization methods can provide query writers with a set of guidelines to write efficient and scalable graph analysis applications. Although the methods proposed here are for the graphs whose the vertices and edges are stored in tables, a primary data structure to store data on the DAS system, they are simple and general enough to be applied to other dataflow architectures.

3.1 Column Reduction

The column reduction basically concerns with reducing the number of columns in tables. This includes constructing virtual tables that contain only those columns that are needed for processing a given query. This also involves generating intermediate results that contain only those columns that will be used in subsequent operations. The column reduction obviously improves the performance of user queries by reducing the volume of data to be stored in disks, reducing the disk I/O time on each local node. More importantly, the reduction in the amount of the data to be processed can significantly reduce data transfers between nodes in a distributed environment and thus potentially expensive inter-node communication time.

3.2 Row Reduction

The row reduction concerns with reducing the number of rows in the tables. The rows in a table can be reduced in a number of different ways. The first method is to eliminate any duplicate rows. In fact, this approach works better if it is applied after the column reduction optimization, because rows that contain different column values in a table may become identical after removing some of the columns. Another viable and effective approach that can be used when a given query has certain constraint conditions is to filter out the rows that do not satisfy the constraints. To be fully effective, the constraint-based row reduction should to be performed in prior to the execution of (usually high-cost) data manipulation operation such as `JOIN` that the constraints are part of. Alternatively, when no constraints are present in the query, data in the tables can be reorganized in such a way that the number of rows in the tables is reduced.

Just as in the column reduction, the row reduction benefits from the reduced I/O and communication time enabled by the reduction in the overall volume of data. The row reduction technique is very effective in optimizing large graph queries, since reducing the number of rows in tables can significantly improve the performance of the `JOIN` operation, which is the most commonly invoked operation by a majority of graph analysis algorithms.

3.3 Data Distribution for Localizing Operation

Data distribution can have a significant impact on the performance and scalability of applications running on any distributed system. The well-balanced distribution of data is even more critical to scaling graph queries to very large graphs on a distributed parallel machine, because the load imbalance created by a data skew can make overloaded nodes to dominate the overall performance.

With this optimization technique, we extend the idea of data distribution further. The idea here is to distribute the data in such a way that not only the load imposed on node is relatively well-balanced, but also subsequent operations can be performed using only the local data stored in each node, eliminating the need for costly inter-node transmission of potentially large volume of data. The distribution of the data to enable the localized operation while assuring the correctness of the queries requires thorough understanding of data. Further, it may be necessary to redistribute the intermediate result sets continuously in order to keep the operations to be performed on the local data. However, it has been observed that the performance gain obtained by the redistribution of data for enabling the localized data manipulation operations outweighs the redistribution cost for most queries.

In case when the cost of distributing entire data is larger than the performance gain, we can still benefit from the distribution of data using some of the light weight operations. A classic use case of the light weight operation is when joining a small table with a much larger table, where light weight join first copying the small table to all the nodes where the large table is stored and then performs local joins. That is, having a local copy of the small table available on each node allows avoiding the transmission of usually large intermediate results and can improve the performance. Sorting data before executing certain operations is another good example of light weight operation.

3.4 Data Preparation

Optimization via data preparation refers to reorganizing data a priori to the execution of a query to ensure the fast execution of the query. The restructuring can involve adding, deleting, and combining the columns and rows of existing tables. In one extreme, we can construct a hierarchical database by merging set of tables into a single table that contains the combined data in a hierarchal form. The resulting hierarchical representation of the graph can reduce query response time drastically for certain types of graph queries than the tabular form commonly used by relational databases.

In most cases, however, the optimization via data preparation is practical only when the target queries are known in advance or the tables are small, since dynamically restructuring large data set can be costly. Nevertheless, preparing data in a form that is ideal for target queries is a viable and effective optimization technique. Preparing a graph in adjacent list stored in a hierarchical form for graph searches is a good example of this optimization technique.

4 Results from Performance Studies

We have measured the effectiveness of the query optimization techniques for dataflow systems. The performance of a suite of benchmark queries that are optimized by applying some of the optimization methods is measured and compared to that of a state-of-art parallel relational database machine. The results are reported in this section.

4.1 Experimental Environment

A real-world semantic graph constructed from the PubMed data [20], which contains information on articles published in medical journals such as title, authors, keywords, and abstract, is used in this performance study. The raw PubMed data is scanned first to extract the entities and the relationship between the entities, and then the semantic graph is created as defined by a given ontology. The constructed PubMed graph has about 30 million vertices and 540 million edges. A subgraph with about 1 million vertices is also prepared by sampling the full-scale PubMed graph for studying the performance of smaller systems.

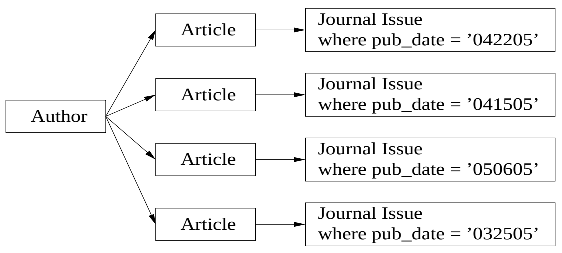
We used a suite of graph queries as benchmarks that were designed to represent a wide spectrum of complex queries [10, 21]. These benchmarks are salient, not only because they represent those queries that information analysts are likely to issue in their data analysis operation but also because they serve as an excellent tool to measure the scalability of a system as they tend to generate a large volume of intermediate results to push the system's capability to its limit.

The benchmarks consist of two types of graph queries: *subgraph pattern matching* and *graph search*. Given a semantic graph and a template graph (also known as a pattern graph), a subgraph pattern matching attempts to find all the instances of subgraphs in the semantic graph that match the given template. Graph search query is used to find paths between two vertices in the graph. The graph benchmarks are depicted in Figure 1.

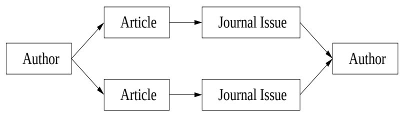
Each pattern query is implemented by first decomposing input pattern into a set of smaller patterns, often in the form of path queries. The instances that match the subqueries are then joined to produce intermediate results that contain those instances of patterns present in the subqueries. The intermediate data (and any resulting from subsequent joins) are repeatedly joined in a hierarchical fashion to find all the instances of subgraphs that match the given pattern. Due to this optimization approach, executing pattern queries invokes a sequence of join operations, many of which are self-joins. Readers should refer [10] for more details on the implementation of the graph pattern queries.

The first query (Query 1) is a highly constrained form of pattern query. This query attempts to find all the authors who published articles in four specific dates. This query is first broken into four smaller patterns, each of which represents the instances of an author-to-date path. The four sets of instances are joined later to find authors who are found in the all four groups. The second query (Query 2) shown in Figure 1.b finds authors who have published two articles in the same journal. Similarly to the Query 1, all the instances of authors who have published any articles in any journals are detected first and then a self-join is performed to find a set of authors who are found in the both sets of instances. Figure 1.c depicts a pattern for the authors who have published four articles in a journal called *Physical Review Letters* (Query 3). As the first step, a path query for all the authors who have published articles in the particular journal is performed. Successive joins of intermediate results will eventually return a set of authors who published more than four papers in the journal. Query 4 shown in Figure 1.d finds two authors who co-authored two or more papers. For this query, all the instances of two authors who co-authored any papers are identified first followed by a self-join. Figure 1.e presents Query 5 that represents sophisticated queries in which specific types of edges are excluded. Here, the Query 5 finds the instances of two articles that do and do not have associated grants, respectively.

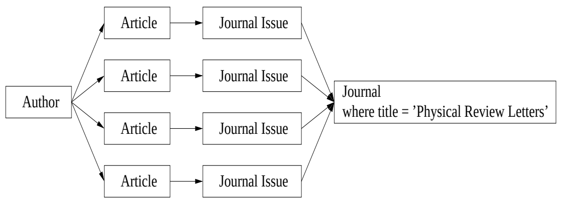
The graph search queries used in the experiments are based on conventional breadth-first search (BFS) algorithm. Though simple, the BFS algorithm is an important benchmark, because the BFS is a fundamental algorithm that is used by a wide range of common graph mining algorithms and the execution of BFS algorithm


Author
Article
Article
Article
Article
Journal Issue where pub_date = '042205'
Journal Issue where pub_date = '041505'
Journal Issue where pub_date = '050605'
Journal Issue where pub_date = '032505'

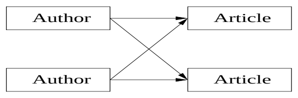
(a) Query 1



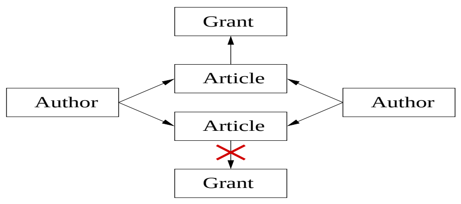
(b) Query 2



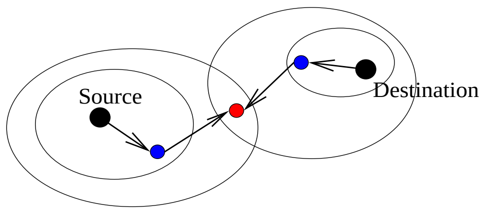
(c) Query 3



(d) Query 4



(e) Query 5



(f) Bidirectional BFS

Figure 1: A suite of graph benchmarks used in the performance study. It consists of five pattern queries and two search queries

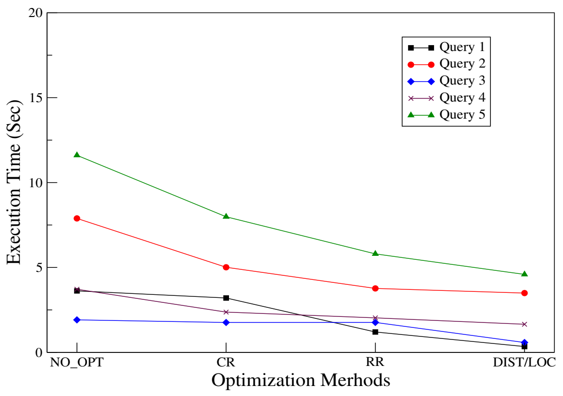
exhibits random memory access behavior that is shared by majority of graph algorithms. Two types of BFS algorithms are examined in this paper: uni- and bi-directional BFS.

The uni-directional BFS is the ordinary BFS algorithm, which finds the shortest path between given source and destination vertices. The bi-directional BFS is a variation of the uni-directional BFS, where two separate breadth-first searches start from the source and destination vertices. These searches continue until a common vertex that can be reached from both ends is found or all the vertices in the graph are visited. Naturally, this search is more efficient than the uni-directional search, because the length of search paths is about half of that of the uni-directional search. Figure 1.f describes the bi-directional BFS algorithm pictorially.

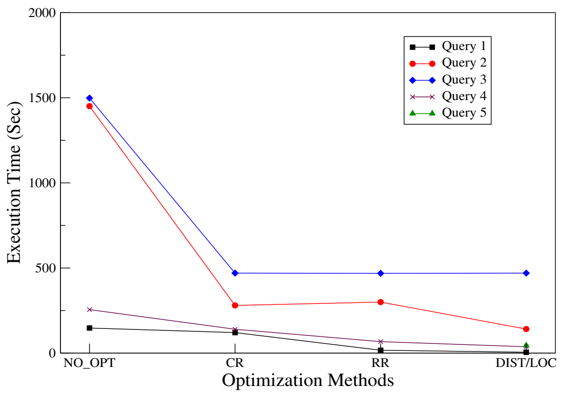
4.2 Performance Evaluation Results

Figure 2 presents the effect of the proposed optimization methods on the query performance. In this study, only three optimization methods, column reduction (CR), row reduction (RR), and distribution/localization (DIST/LOC), are considered mainly because the benchmarks considered here do not have a rich set of constraints of which other proposed optimization techniques can take advantage. The three optimization methods are gradually applied to the baseline (unoptimized) queries in order to evaluate the effect of each optimization technique individually. Hence, the data points for DIST/LOC in the graphs correspond to the performance of fully optimized queries.

As shown in the figure, the simple column reduction can improve the performance significantly. Its impact is more evident for the Queries 2 and 3 in Figure 2.b, as these queries are more complicated and generate larger intermediate results than others. In contrast, Query 1 contains constraints that its baseline implementation can take advantage of to filter out a large number of records prior to join operations, and therefore, the effect of the column reduction is minimal as shown in Figure 2.b. The performance impact of the column reduction for Query



(a) Effect of optimization for small PubMed graph



(b) Effect of optimization for large PubMed graph

Figure 2: Measuring the effect of proposed optimization methods using PubMed graphs. A smaller PubMed graph with 1 million vertices and a full-scale PubMed graph with 30 million vertices are used in this experiment. Each method is gradually applied in optimizing the target queries.

3 is not clear in Figure 2.a, because a smaller PubMed graph is used in the experiment.

The number of rows is reduced in these experiments mainly by creating a separate table for each sub-query so that the sub-queries can be run on smaller tables. In Figure 2, the effect of the row reduction varies to great extent largely depending on the queries, since some queries benefit from this optimization as it offers smaller tables for fast joins, whereas for others the number of rows reduced is too small to have significant performance impact.

A primary goal of the distribution for localization is to reduce the overall communication time spent in join operations by enabling local joins. It was observed that this technique improves the performance of all the queries examined in the experiments, because this technique basically can eliminate any data skews. This optimization technique not only can improve the performance, but sometimes it enables the execution of queries on large data sets. For example, the Query 5 on the 30 million-vertex PubMed graph could not be run on DAS even with the column and row reductions due to the large skew of the data. Elimination of the data skew through data redistribution enabled the execution of the query. It is clear that the effect of the proposed optimization methods varies to great extent depending on the query and data. However, the fact that using any combinations of the proposed methods does not degrade performance as shown in Figure 2 suggests that users should exercise these optimization techniques when writing a query.

Table 1 presents uni- and bi-directional BFS search performance on the DAS system. Full-scale PubMed graph is searched in this experiment. Each search is conducted to find the shortest path between two randomly chosen vertices. The search performance is obtained from the average of 20 searchers. The performance of optimized and unoptimized searches is compared in the table. As Table 1 indicates, the optimized BFS algorithms achieved 2.4X and 3.6X speedups over their unoptimized counterparts. The improvement in the search performance is enabled by optimizing the search algorithms via data preparation. Here, input graph represented as an edge table is rearranged to form an adjacency list, where each entry consists of a source vertex and its adjacent vertices. In a BFS search, an expansion from the current level to the next involves finding a set of vertices that can be reached from the vertices at the current level and requires a self-join of the input edge list. Conversion to adjacency list reduces the amount of data to be joined, resulting in significant reduction in the join time.

The performance of the optimized queries on the dataflow testbed is compared to that of equivalent SQL queries on a single-node Oracle RDBMS server in Table 2. The smaller PubMed graph is used here, due to the limited hardware resources on the Oracle server. Queries for the DAS system are optimized via applying combinations of applicable optimization methods proposed in this paper. As Table 2 shows, the optimized

	Unoptimized	Optimized
Uni-directional BFS	287.926	120.359
Bi-directional BFS	204.902	56.431

Table 1: Performance of uni- and bi-directional BFS searches on the DAS system (in seconds). Full-scale PubMed graph with 30 million vertices is used in this experiment.

	DAS	RDBMS	Speedup/Node
Query 1	0.343	3	0.44
Query 2	3.182	1014	15.93
Query 3	0.573	21.8	1.90
Query 4	1.742	387	11.11
Query 5	4.575	282	3.08

Table 2: Comparing the performance of queries on DAS and Oracle RDBMS for 1 million-vertex PubMed graph (in seconds).

	DAS	Netezza	Speedup/Node
Query 1	9.422	27.3	7.82
Query 2	142.099	834.47	15.86
Query 3	469.511	15392.96	88.52
Query 4	37.803	741.42	52.95
Query 5	44.6	496.48	30.06

Table 3: Comparing the performance of queries on DAS and Oracle RDBMS for 30 million-vertex PubMed graph (in seconds).

queries on the dataflow system outperform corresponding SQL queries on Oracle by orders of magnitude in most cases, achieving as much as 16X speedup per node. Such performance improvement can be attributed to the fact that the optimization of the queries via proposed techniques ensures small intermediate results and minimal communication. Highly optimized underlying query execution engine of the DAS system also contributes to the fast execution of the queries.

The performance of the same set of queries for the large-scale PubMed graph on the DAS system and a 54-node Netezza Performance Server (NPS), a specialized active-disk based distributed relational data management system, is compared in Table 3. The SQL queries are optimized by applying the column reduction for the fairer comparison. We relied on the Netezza's SQL compiler for the rest of optimization, As the table shows, the DAS system achieves almost two orders of magnitude speedup per node over the Netezza system. The performance improvement is most noticeable for the Queries 3 and 4. This is because these queries consider all the possible combinations of intermediate results, easily leading to combinatorial explosion of data. Since the amount of the intermediate data to be joined is reduced to a great extent by the distribution/localization method for the dataflow queries, the negative effect of the combinatorial explosion is small on DAS.

5 Concluding Remarks

In this work, the feasibility of using dataflow systems as a large-scale semantic graph analysis engine is studied. A general query optimization framework for parallel dataflow systems is also proposed and applied to optimizing a suite of graph query benchmarks. The performance of the optimized queries measured on an actual parallel dataflow system using a large-scale semantic graph validates the effectiveness of the proposed techniques. Furthermore, the comparison of the optimized query performance to that of equivalent SQL queries on a high-end parallel relational database machine shows that we can achieve orders of magnitude improvement in query performance on dataflow systems over state-of-art relational database systems, verifying that the dataflow machines can be a viable and scalable alternative to run large complex graph queries.

Acknowledgements

Author would like to thank Jim Morton for his help managing the DAS system and David Bayliss and Jim Wiltshire for their insightful comments on query optimization on DAS.

References

[1] D. Bayliss, R. Chapman, J. Smith, O. Poulsen, G. Halliday, and N. Hicks. Query scheduling in a parallel-processing database system, 2007. US Patent 7185003.

[2] D. Bayliss, R. Chapman, J. Smith, O. Poulsen, G. Halliday, and N. Hicks. System and method for configuring a parallel-processing database system, 2007. US Patent 7240059.

[3] H. Blau, N. Immerman, and D. Jensen. A visual language for querying and updating graphs. Technical Report University of Massachusetts Amherst, Computer Science Department Technical Report 2002-037, University of Massachusetts Amherst, 2002.

[4] D. Chakrabarti. Autopart: parameter-free graph partitioning and outlier detection. In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 112–124, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[5] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[6] J. B. Dennis and G. R. Gao. An efficient pipelined dataflow processor architecture. In *Supercomputing '88: Proceedings of the 1988 ACM/IEEE conference on Supercomputing*, pages 368–373, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.

[7] J. B. Dennis and D. P. Misunas. A preliminary architecture for a basic data-flow processor. In *ISCA '98: 25 years of the international symposia on Computer architecture (selected papers)*, pages 125–131, New York, NY, USA, 1998. ACM.

[8] G. Estrin and R. Turn. Automatic assignment of computations in a variable structure computer system. *IEEE Trans. Electronic Computers*, EC-12(5):755–773, 1963.

[9] I. Kaplan. A semantic graph query language. Technical Report UCRL-TR-255447, Lawrence Livermore National Laboratory, 2006.

[10] I. Kaplan, G. Abdulla, T. Brugger, and S. R. Kohn. Implementing graph pattern queries on a relational database. Technical Report LLNL-TR-400310, Lawrence Livermore National Laboratory, 2008.

[11] R. M. Karp and R. E. Miller. Parallel program schemata: A mathematical model for parallel computation. In *FOCS*, pages 55–61, 1967.

[12] Lexis-Nexis Data Analytics Supercomputer. `http://www.lexisnexis.com`.

[13] W. A. Najjar, E. A. Lee, and G. R. Gao. Advances in the dataflow computational model. *Parallel Comput.*, 25(13-14):1907–1929, 1999.

[14] M. E. J. Newman. Detecting community structure in networks. *European Physical Journal B*, 38:321–330, May 2004.

[15] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69(6):066133, June 2004.

[16] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb. 2004.

[17] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814, 2005.

[18] E. Riedel, C. Faloutsos, G. A. Gibson, and D. Nagle. Active disks for large-scale data processing. *IEEE Computer*, June 2001.

[19] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[20] U.S. National Institutes of Health (NIH). `www.pubmedcentral.nih.gov`.

[21] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, and Ümit Çatalyürek. A scalable distributed parallel breadth-first search algorithm on bluegene/l. In *Proceedings of Supercomputing'05*, Nov. 2005.